Large Margin Vector Quantization.

Lawrence Buckingham and Shlomo Geva
{l.buckingham|s.geva}@qut.edu.au
Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia

Abstract

In this paper we describe the Large Margin Vector Quantization algorithm (LMVQ), which uses gradient ascent to maximise the margin of a radial basis function classifier. We present a derivation of the algorithm, which proceeds from an estimate of the class-conditional probability densities. We show that the key behaviour of Kohonen's well-known LVQ2 and LVQ3 algorithms emerge as natural consequences of our formulation. We compare the performance of LMVQ with that of Kohonen's LVQ algorithms on an artificial classification problem and several well known benchmark classification tasks. We find that the classifiers produced by LMVQ attain a level of accuracy that compares well with those obtained via LVQ1, LVQ2 and LVQ3, with reduced storage complexity. We indicate future directions of enquiry based on the large margin approach to Learning Vector Quantization.

1 Introduction

Optimal margin classifiers have recently become prominent in the search for improved classification algorithms. The margin was originally conceived as a measure of the separation between correctly classified training examples and the decision surface of a classifier, arising from the principles of Structural Risk Minimisation [Vapnik 1995]. The idea of maximising the margin forms the basis for the Support Vector Machine family of inductive learning algorithms [Vapnik 1995, Boser, Guyon & Vapnik 1992], which maximise the minimum margin over the training set. Following empirical successes enjoyed by boosting algorithms, in particular AdaBoost [Freund & Schapire 1997], it has been established that several boosting algorithms maximise a function of the margin over the training set.

The general idea behind the theory of optimal margin classifiers is that, if a classifier is able to correctly classify a large portion of the training set with wide separation from the decision surface, then one may expect to see good generalisa-

tion performance from the classifier. A growing body of theoretical results support this idea. [Shawe-Taylor *et al.* 1996] prove a bound for the generalisation error of maximum margin hyperplanes in terms of the error over the training set and the margin. Several authors demonstrate that AdaBoost performs gradient ascent maximisation of the margin over the training set [Friedman *et al* 1998, Mason *et al.* 1999, Breiman 1997], while [Shapire *et al.* 1998] provide a bound for the expected generalisation error of voting ensemble classifiers, including those produced by AdaBoost.

Meanwhile, the Learning Vector Quantization (LVQ) [Kohonen 1997] family of algorithms enjoy sustained popularity. The LVQ algorithms are efficient procedures for constructing nearest neighbour classifiers from a set of examples. They have been applied widely in many domains including pattern recognition, speech recognition and signal processing.

As originally formulated, the LVQ algorithms can be shown to be approximate attempts to model the point-density of a function which is similar to the margin [Kohonen 1997, pages 203–207]. Kohonen derives an "ideal" training algorithm which is defined in terms of a set of optimal discriminant functions. Unfortunately, the optimal discriminant functions are not known in the general case. Each LVQ variant is therefore a heuristic approximation to this ideal algorithm.

In the present paper, we derive a gradient training algorithm for a family of normalised radial basis function classifiers, which we call Large Margin Vector Quantization (LMVQ). Whereas standard LVQ is based on hard partitioning of the input space, using the 1-nearest neighbour criterion, LMVQ forms a soft partition of the input space. We show that the widely known LVQ1, LVQ2.1 and LVQ3 algorithms are discrete approximations to this algorithm, each of which exhibits an aspect of the behaviour of LMVQ. In this sense, LMVQ unifies the LVQ algorithms.

The paper is organised as follows. Section 2 introduces the learning problem and provides definitions of the terms involved in our discussion. In Section 3 we briefly review the basic concepts that underlie LVQ, the theory backing the idea and describe the main flavours of LVQ. This is done in order to provide a context and points of comparison for the qualitative discussion in the next section. An abstract training algorithm is derived in Section 4, cast in terms of an arbitrary set of basis functions. This allows us to conduct qualitative analysis of the general properties of the algorithm, comparing and contrasting LMVQ with LVQ algorithms. In Section 5, we produce a concrete training algorithm which uses radially symmetric gaussian kernels. Sections 6 and 7 set out the results of empirical tests which confirm the feasibility and utility of the LMVQ algorithm, while Section 8 presents our conclusions.

2 The classification problem

We consider classification problems, in which there is an input set $\mathcal{X} \subset \Re^D$, a discrete set of class labels $\mathcal{C} = \{1, ..., C\}$, and an unknown relation $\mathcal{R} \subset \mathcal{X} \times \mathcal{C}$. We have a (given) training set of labelled examples $\mathcal{T} \subset \mathcal{R}$, which are assumed to be drawn according to some fixed but unknown probability distribution $\mathcal{D}$ defined over $\mathcal{R}$. The goal is to find a function $F : \mathcal{X} \to \mathcal{C}$ which approximates $\mathcal{R}$ in such a way that the probability of classification error is minimal with respect to $\mathcal{D}$.

Given an observation $(x, y) \in \mathcal{R}$, we take the following definitions:

- $p(x)$ is the density governing the probability of drawing a sample with input value x.
- $P(y = c)$ is the prior probability that the class label y takes the value c.
- $p(x|y = c)$ is the density governing the conditional probability that x is drawn, given that the class y is c.
- $P(y = c|x)$ is the probability that class label c is drawn, given that the input is x.

According to Bayes' Rule, these quantities are related by the expression:

$$P(y = c|x)p(x) = p(x|y = c)P(y = c). \tag{1}$$

Taken together over all classes $c \in \mathcal{C}$, (1) gives rise to a family of discriminant functions which may be used to predict the class label for an observation. This classifier, which minimises the expected number of classification errors with respect to $\mathcal{D}$, will be called the *Bayes classifier*:

$$\begin{aligned} F(x) &= \arg\max_c \{f_c(x)\}, \\ \text{where } f_c(x) &= \frac{p(x|y = c)P(y = c)}{p(x)}, \\ \text{for } c &= 1, ..., C. \end{aligned} \tag{2}$$

Following [Shapire *et al.* 1998], we define the *margin* of an observation (x, y) as:

$$margin(x, y) = f_y(x) - \max_{c \neq y} \{f_c(x)\}. \tag{3}$$

Since $0 \leq f_c(x) \leq 1$ for each class c and all $x \in \mathcal{X}$, it follows that $-1 \leq margin(x, y) \leq 1$. A positive value of the margin implies that the predicted class matches the observed class, with a value close to 1 capturing a sense of "confident correct classification".

It has been observed that the maximum term appearing on the right hand side of (3) is awkward to work with [Breiman 1997]. Breiman defines a quantity called the *edge* which resembles the margin but features a summation in place of the maximum term:

$$edge(x, y) = f_y(x) - \sum_{c \neq y} f_c(x). \tag{4}$$

Since all terms in in the summation are non-negative, the edge is a lower bound on the margin and in the special case where $\mathcal{C}$ consists of two classes, the edge becomes identical to the margin. Further, since we know that for all $x \in \mathcal{X}$, $\sum_{c=1}^{C} f_c(x) = 1$, we have

$$edge(x, y) = 2f_y(x) - 1. \tag{5}$$

3 Review of Learning Vector Quantization Algorithms

The term LVQ is commonly used to refer to a family of algorithms which construct nearest neighbour classifiers from sets of labelled examples. In a nearest neighbour classifier we create a *codebook*, consisting of a set of prototypes belonging to $\mathcal{X} \times \mathcal{C}$. The codebook is a set of $M \geq C$ labelled pairs, $\mathcal{M} = \{(\mu_1, \phi_1), ..., (\mu_M, \phi_M)\} \subset \mathcal{X} \times \mathcal{C}$.[1] The matter of determining the optimal number of prototypes to place in the codebook is an open problem.

Initially a codebook is created by drawing random observations from the training set. Then the codebook is optimised by applying one or more of the LVQ training algorithms described below.

The basic Learning Vector Quantization process [Kohonen 1997, page 205] is an iterated on-line supervised learning step. At each time step t, observation $(x(t), y(t))$ is drawn from the training set and a decision is made to determine which (if any) prototypes must be updated in the current time step. The general form of Kohonen's LVQ algorithms is as follows:

$$\mu \quad \leftarrow \quad \mu \pm \alpha(t) m(x(t), \mu)(x(t) - \mu). \tag{6}$$

The update is positive if and only if $y(t)$ is the same as ϕ, that is, the classes of prototype and training example match. The learning step size $\alpha(t)$ may change as training progresses. The learning modifier $m(x(t), \mu)$ is introduced to encapsulate the decision process that determines which, if any, training examples are of interest when a particular prototype is considered for update.

[1] On occasions in the ensuing discussion we refer to the nearness of a training observation to a prototype. This should be taken to mean the similarity between the input value of the training observation and the reference pattern associated with the prototype.

There are 3 main variants of LVQ, each of which represents a heuristic attempt to maximise a function of the optimal Bayes classifier which is very similar to the margin of the Bayes classifier. At time step t, let $\{\mu_i, \mu_j\}$ be the nearest and second-nearest neighbours of $x(t)$ in the codebook, at distances $\{d_i, d_j\}$ from $x(t)$, having classes $\{\phi_i, \phi_j\}$ respectively. Then each of Kohonen's LVQ algorithms is given by a different choice of the learning modifier, as set out below.

$$\text{LVQ1: } m(x(t),\mu) = \begin{cases} 1 & \text{if} \quad \mu = \mu_i \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

$$\text{LVQ2.1: } m(x(t),\mu) = \begin{cases} 1 & \text{if} \quad \mu \in \{\mu_i, \mu_j\}\,,\ \phi_i \neq \phi_j, \text{ and} \\ & \quad \phi_i = y(t) \text{ or } \phi_j = y(t), \text{ and} \\ & \quad \min\left(d_i/d_j, d_j/d_i\right) > \frac{1-w}{1+w}, 0 < w < 1. \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

$$\text{LVQ3: } m(x(t),\mu) = \begin{cases} & 1 \quad \text{if} \quad \mu \in \{\mu_i, \mu_j\}\,,\ \phi_i \neq \phi_j, \text{ and} \\ & \quad \phi_i = y(t) \text{ or } \phi_j = y(t), \text{ and} \\ & \quad \min\left(d_i/d_j, d_j/d_i\right) > \frac{1-w}{1+w}, 0 < w < 1. \\ \epsilon \in (0, 0.5) & \text{if} \quad \mu \in \{\mu_i, \mu_j\}\,, \text{ and} \\ & \quad \phi_i = \phi_j = y(t), \text{ and} \\ & \quad \min\left(d_i/d_j, d_j/d_i\right) > \frac{1-w}{1+w}, 0 < w < 1. \\ & 0 \quad \text{otherwise.} \end{cases} \tag{9}$$

The LVQ1 rule is very conservative, primarily modelling the point density of each class with the prototypes assigned to that class. The motivation for LVQ2.1 is to improve the estimate of the decision surface, by adjusting pairs of prototypes that seem to straddle the decision surface. However, the LVQ2.1 rule works in such a way that prototypes tend to be drawn toward the decision surface. If the prototypes approach too closely, training can become unstable, since a small change to a prototype may cause a large change in the decision regions of the classifer. LVQ3 attempts to compensate for these difficulties by introducing a restoring force that pulls prototypes back toward examples with matching class.

We shall see that these issues are addressed in a very natural manner by the LMVQ algorithm. The behaviour of LMVQ is governed by the relative strengths of estimated conditional probability densities. In some situations LMVQ behaves like LVQ2.1, while in others it emulates LVQ3.

4 The Abstract LMVQ Algorithm

In this section and the sequel we derive a training algorithm that maximises the margin of a normalised radial basis function classifier [Geva & Sitte 1991b, Jordan & Xu 1995]. The starting point for the derivation is the Bayes classifier given by (2), which we approximate by estimating the true class-conditional

densities with a weighted sum of basis functions. Substituting the resulting approximate discriminant functions into (5) gives the edge of each observation, which we then maximise by gradient ascent to obtain the LMVQ learning step.

Let $\mathcal{G}$ be a set of basis functions $\{g(\cdot;\theta) : \mathcal{X} \to \Re\}$, where $\theta \in \Re^K$ is a vector of real-valued parameters which can be used to select a specific member of $\mathcal{G}$. We assume that each $g \in \mathcal{G}$ is differentiable with respect all components of θ, and that it satisfies the constraints:

$$\begin{aligned} \forall x \in \mathcal{X}, \quad g(x) &\geq 0, \\ \int_{\mathcal{X}} g(x)\mathrm{d}x &= 1. \end{aligned} \tag{10}$$

By choosing a set of specific values for $\{\theta_i\}_{i=1}^M$ we obtain a parametrised set of basis functions $\{g_i(\cdot)\}_{i=1}^M$. We assign each parameter vector θ_i to a class by associating it with a class label $\phi_i \in C$. Writing the density $p(x)$ as a weighted sum of basis functions gives:

$$p(x) = \sum_{i=1}^{M} w_i g_i(x), \text{ and} \tag{11}$$

$$p(x|y=c) = \sum_{i=1}^{M} \delta_{\phi_i c} w_i g_i(x), \tag{12}$$

$$f_c(x) = \frac{\sum_{i=1}^{M} \delta_{\phi_i c} w_i g_i(x)}{\sum_{i=1}^{M} w_i g_i(x)} \tag{13}$$

Here δ_{ij} is the Kronecker delta function,

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Substituting f_c from (13) into (2) defines the abstract LMVQ classifier,

$$F(x) = \arg\max_c \left\{ \frac{\sum_{i=1}^{M} \delta_{\phi_i c} w_i g_i(x)}{\sum_{i=1}^{M} w_i g_i(x)} \right\} \tag{14}$$

Our goal is to adjust the parameters $\{\theta_i\}_{i=1}^M$ to try and improve the average value of $f_y(x)$ taken over $\mathcal{T}$, and by so doing improve the average margin of examples in the training set.

Let θ_{ij} be one component of parameter vector θ_i. Gradient ascent on the value of f_y calls for an iteration of the form

$$\theta_{ij} \leftarrow \theta_{ij} + \rho \frac{\partial f_y}{\partial \theta_{ij}} \tag{15}$$

where $\rho > 0$ is a (usually small) step size, which may vary with time.

Differentiating f_y with respect to θ_{ij}, and noting that $\partial g_k / \partial \theta_{ij} = 0$ if $k \neq i$, we obtain the following abstract on-line learning step:

$$\theta_{ij} \quad \leftarrow \quad \theta_{ij} + \frac{\rho w_i}{\left(\sum_{k=1}^{M} w_k g_k(x)\right)^2} \frac{\partial g_i}{\partial \theta_{ij}} \left(\sum_{k=1}^{M} w_k g_k(x) \left(\delta_{\phi_i y} - \delta_{\phi_k y}\right) \right) \tag{16}$$

To obtain a concrete training algorithm all we need to do is choose a particular class of basis functions $\mathcal{G}$ and substitute the values of g_k and $\partial g_i / \partial \theta_{ij}$. However, if we break down (16) further, it is possible to determine which basis functions play an active rôle and what will be the qualitative effect of each learning step.

The sign of the right-hand side of (16) determines whether a "reward" or "punishment" occurs. This is determined by the sign of

$$R \quad = \quad \sum_{k=1}^{M} w_k g_k(x) \left(\delta_{\phi_i y} - \delta_{\phi_k y}\right). \tag{17}$$

Splitting this summation into two parts gives

$$R \quad = \quad \sum_{\phi_k = \phi_i} w_k g_k(x) \left(\delta_{\phi_i y} - \delta_{\phi_k y}\right) \tag{18}$$

$$+ \quad \sum_{\phi_k \neq \phi_i} w_k g_k(x) \left(\delta_{\phi_i y} - \delta_{\phi_k y}\right). \tag{19}$$

If $\phi_k = \phi_i$ then the quantity $(\delta_{\phi_i y} - \delta_{\phi_k y})$ is identically zero, so the first part of the summation (line 18) disappears. However, in line 19 there are two possibilities:

1. $y = \phi_i$. This implies that $y \neq \phi_k$, so we have

$$R \quad = \quad \sum_{\phi_k \neq y} w_k g_k(x) \quad \geq 0. \tag{20}$$

   Here, all basis functions allocated to classes that *do not match* that of the training observation are involved in a "reward" step applied to θ_i.

2. $y \neq \phi_i$. In this case,

$$R \quad = \quad -\sum_{\phi_k = y} w_k g_k(x) \quad \leq 0. \tag{21}$$

   That is, a "punishment" step is applied to θ_i, involving only those basis functions allocated to *the same* class as the current training observation.

We see that a reward-punishment regime similar to that of Kohonen's LVQ algorithms will emerge for a wide choice of basis functions $\mathcal{G}$ satisfying the constraints imposed by (10).

5 LMVQ Training Algorithm

In this section we turn to a concrete choice of basis functions, derive an explicit learning step, and compare the resulting algorithm to the LVQ algorithms described in Section 3.

Consider the family of radially symmetric multivariate gaussian kernels, with centres at $\mu_i \in \Re^D$ and bandwidth $\sigma \in \Re$.[2]

Formally replacing θ_i with the pair $(\mu_i^T, \sigma)^T$, let

$$\begin{aligned} g_i(x) &= g(x, \mu_i, \sigma) \\ &= \frac{1}{\sqrt{2\pi\sigma^D}} \exp\left(-\frac{\|x-\mu_i\|^2}{2\sigma^2}\right). \end{aligned} \tag{22}$$

Letting $\mathcal{M} = \{(\mu_i, \phi_i); i = 1, ..., M\}$ and allowing σ to approach 0 yields the familiar 1-nearest neighbour classifier. Alternatively, if $\sigma > 0$ then the kernel functions produce a soft partition of the input space and the classifier uses a weighted vote to choose the predicted class. Either way, $\mathcal{M}$ can be thought of as a generalised codebook.

To obtain the final LMVQ learning step, differentiate g_i with respect to component j of vector μ_i and substitute into (16) to obtain

$$\mu_{ij} \leftarrow \mu_{ij} + \frac{\rho w_i g_i(x) \left(\sum_{k=1}^M w_k g_k(x) \left(\delta_{\phi_i y} - \delta_{\phi_k y}\right)\right)}{\sigma^2 \left(\sum_{k=1}^M w_k g_k(x)\right)^2} (x_j - \mu_{ij}) \tag{23}$$

Equation (23) takes the same form as the LVQ learning step 6, preserving that prototypes with matching class are attracted while those with non-matching class are repelled. However, we see the following differences.

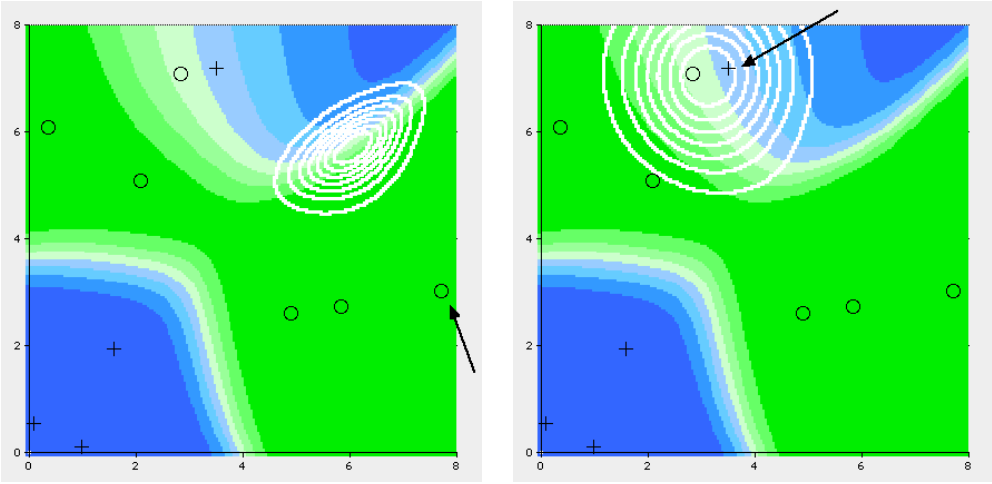
1. Soft competition between prototypes allows an arbitrary number of prototypes to be adjusted for every training observation.

2. The learning modifier is a smooth function rather than a step function.

$$m_i(x) = \frac{g_i(x)}{p(x)} \frac{\sum_{k=1}^M w_k g_k(x) \left(\delta_{\phi_i y} - \delta_{\phi_k y}\right)}{p(x)} \tag{24}$$

3. LMVQ tries to move prototypes away from the estimated decision surface. This increases the expected margin of examples near the decision surface, and also tends to smooth the decision surface. Unlike LVQ1, no specific attempt is made to model the distribution of training examples.

[2] Here we consider a fixed bandwidth σ, shared by all kernels. This approach is widely used for radial basis function networks [Bugmann 1998]. However, it is possible to assign a distinct bandwidth parameter to each kernel. This is one aspect that we intend to explore in future research.



(a) (b)

Figure 1: An artificial LMVQ classifier. (a) LVQ2.1 behaviour emerges when prototypes are relatively far from the estimated decision surface. (b) LVQ3 behaviour is manifested when prototypes are very close to the estimated decision surface

The LMVQ learning modifier $m_i(x)$ effectively filters the training set, allowing an update to μ_i only if the kernel centred on μ_i makes a substantial contribution to the classification decision at x. Also, one of the following conditions must be met for the example to have a measurable effect on the prototype:

- if the observed class matches that of the prototype, then the combined vote of all kernels belonging to *other classes* must be significant at x; while
- if the observed class does not match that of the prototype, then the combined effect of kernels belonging to *the observed class* must be substantial at x.

Taken together, these constraints have the effect of selecting examples for update only if they fall in a neighbourhood of the current estimate of the decision surface and the current prototype. The size of this neighbourhood is determined by the nearness of the prototype to the estimated decision surface and also by the bandwidth σ.

Figure 1 depicts two views of a LMVQ classifier, representing an artificial 2-class problem (generalised XOR). Prototypes were selected at random, and

plotted with markers to indicate their class (X or O). The shading shows contours of the function $f_1(x) - f_2(x)$, while the white curves show contours of the learning modifier generated by one prototype (indicated in each figure by an arrow). It is easy to see the behaviour of both LVQ2.1 and LVQ3 emerging. In Figure 1(a), the prototype under consideration is a long way from the decision surface of the classifier, so that a window effect like that of LVQ2.1 arises. In Figure 1(b), the prototype under consideration is very close to the decision surface of the classifier, causing the "window" to spread out beyond the prototype, giving a restoring effect as embodied in LVQ3.

The LVQ algorithms described in Section 3 can be viewed as discrete approximations to LMVQ. As indicated above, it is easy to see behaviour like that of LVQ2.1 and LVQ3 emerging from the LMVQ learning modifier. By assuming that classification is dominated by the nearest prototype and discarding the constraint on the combined votes of other kernels, we recover LVQ1 as a coarse approximation to LMVQ. In fact, LVQ1 can be shown to arise more naturally if one maximises the margin of an unnormalised radial basis function network.

In more complicated situations with a greater number of prototypes, the local maxima of $m_i(x)$ seem likely to lie between pairs of nearby prototypes since the influence of more distant kernels drops away exponentially. Since the important training examples are always chosen from an area near the current estimate of the decision surface, LMVQ tends to produce a model of the decision surface rather than the distribution of training examples.

6 Experiments on an artificial problem

This section describes a series of experiments we conducted to establish whether the LMVQ algorithm is a practical training algorithm. In these tests we compare the performance of LMVQ with that of LVQ1, LVQ2.1 and LVQ3, using an artificial domain which was designed to clearly show the points of comparison between the algorithms. Reference implementations of the LVQ algorithms, as provided in LVQPAK [3] were used.

The problem consists of samples drawn from two classes, generated from the uniform distribution on the unit square. Examples were classified according to the rule:

$$y = \begin{cases} 1 & \text{if } \|x - (0.5, 0.5)^T\| < 1/\sqrt{2\pi} \\ 2 & otherwise \end{cases} \tag{25}$$

That is, points on the interior of a circle with radius $1/\sqrt{2\pi}$ are labelled as class 1. Both classes appear with approximately equal probability in the training set. Independent training and test sets were generated. After the examples were generated, controlled noise was introduced into the training set by flipping the

[3] Available via anonymous ftp at `cochlea.hut.fi`

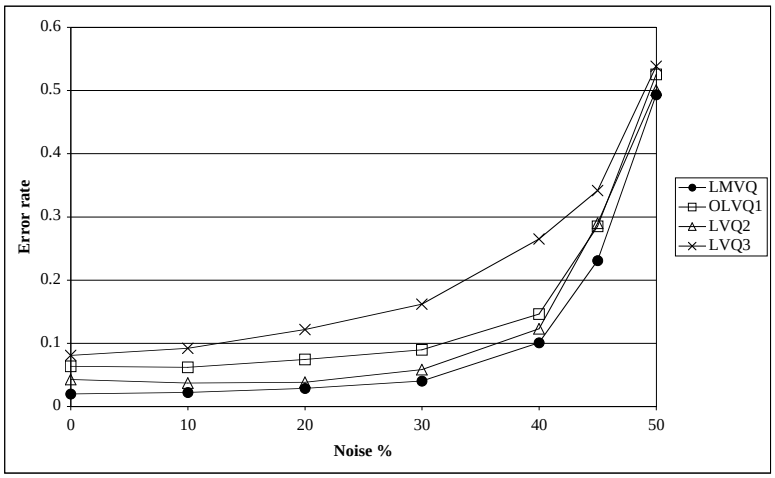


Figure 2: Comparison between error rate of LMVQ, OLVQ1, LVQ2.1 and LVQ3 on an artificial classification problem.

class labels of a fixed proportion of randomly chosen examples. Label noise was *not* introduced into the test set, as we wished to see how well the target concept is uncovered by each classifier.

Altogether, 7 experiments were conducted, corresponding to introduced noise levels of $N = 0\%$, 10%, 20%, 30%, 40%, 45% and 50% respectively. Each experiment consisted of 10 repeats of the following sequence of actions:

1. Generate 2000 training and 2000 test examples.
2. Flip class of N% of training examples.
3. Select initial codebook containing $M = 20$ prototypes from training set, using facilities provided by LVQPAK to initialise the codebook. Separate copies of this codebook were used to construct each classifier.
4. Construct an LMVQ classifier, using fixed bandwidth of 0.1, step size of 0.05, presenting each example 20 times.
5. The Optimised LVQ1 (OLVQ1) algorithm was executed for 10 presentations of each training example, to create an intermediate codebook, which was used as the starting point for the three LVQ classifiers.
6. Each of OLVQ1, LVQ2.1 and LVQ3 were executed for a further 10 presentations of each example. The window size was set to 0.3 for LVQ2.1 and LVQ3. The restoring step size ϵ was set to 0.5 for LVQ3.

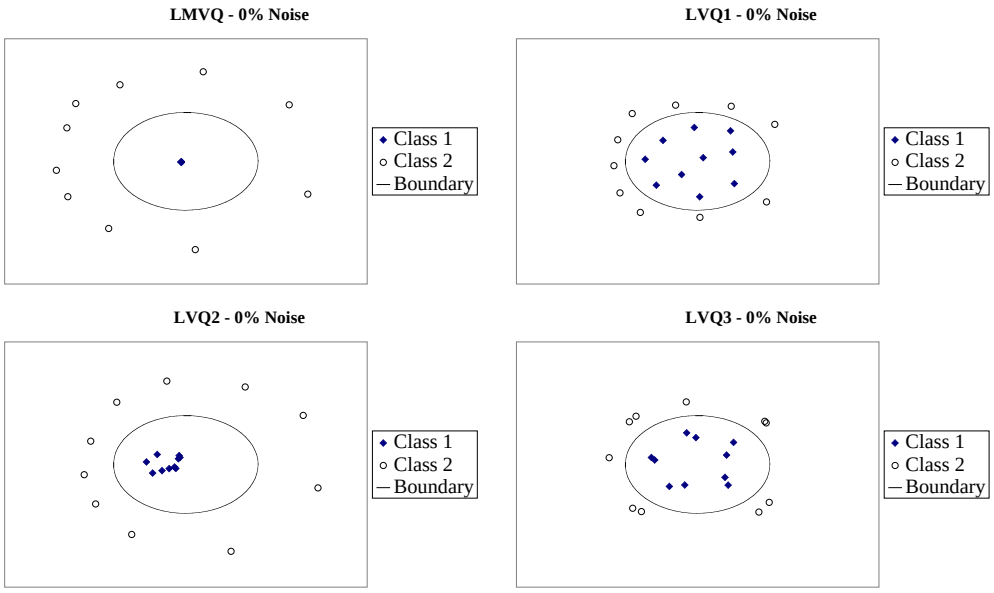


Figure 3: Codebooks produced by the 4 classifiers from one of the experiments, in which no label noise was introduced.

The results of the experiment are presented in Figure 2, which depicts the average error rate, computed as (number of misclassified test examples)/(size of test set). Error bars have been omitted to permit clear presentation of the information. The performance of all algorithms deteriorates gracefully in the presence of moderate class label noise. However, LMVQ consistently produces the most accurate classifiers for this task.

Figure 3 shows the final locations of prototypes after training with each algorithm, for one repeat of the experiment, using training data with no added class label noise. For reference purposes, the true decision surface used to generate examples is also plotted. Looking at the distribution of prototypes within the circle, we see that two extremes of behaviour are exhibited by LVQ1 and LMVQ. While LVQ1 distributes prototypes evenly throughout the region, LMVQ moves all prototypes to the centre of the circle. While this is undoubtedly due to the symmetry of the problem, the difference is striking. In a practical implementation of LMVQ, when prototypes collapse onto the same location in this way the duplicates could be removed, permitting a more economical classifier with no loss of accuracy. This is an economy that is not possible with LVQ1. As might be expected, LVQ2 and LVQ3 exhibit behaviour intermediate between these two extremes.

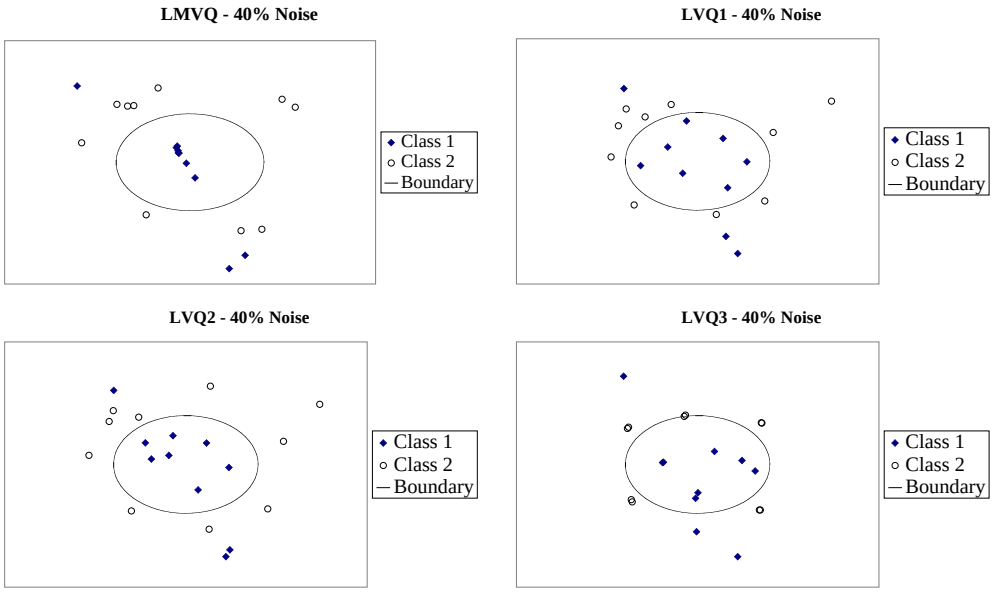


Figure 4: Codebooks produced by the 4 classifiers from one of the experiments, in which 40% label noise was introduced.

All algorithms suffer once large amounts of label noise appear in the training set. This situation is shown in Figure 4, where we have plotted the final locations of prototypes after training with noisy data. Label noise takes a heavy toll on LVQ1 and LVQ2.1, while LMVQ manages to keep most prototypes in the right general area.

One aspect that is not illustrated by these final codebook snapshots is the stability of the LMVQ algorithm. The algorithm readily moves prototypes into stable patterns, where the interaction introduced via the learning modifier serves to lock them in place. The main factor that determines when this will occur is the value of the bandwidth σ, which controls the resolution at which LMVQ models the decision surface. A large value of σ permits the mapping of coarse features only, while smaller values of σ allow correspondingly more detailed mapping of the decision regions.

7 Experiments on benchmark data sets

To see how LMVQ compares with the LVQ algorithms on more complex learning tasks, we conducted a set of experiments with 6 classification benchmark data

Table 1: Benchmark data sets.

Name	Records	Attributes	Classes
Wisconsin Breast Cancer	699	7	2
Glass	214	9	6
Cleveland Heart Disease	303	13	2
Iris	150	4	3
Pima Indian Diabetes	768	8	2
Sonar	208	55	2

sets obtained from the UCI Machine Learning Repository [Murphy & Aha 1994]. Table 1 shows a brief summary of the properties of these data sets.

For each data set, we conducted 10 repeats of 10-fold cross validation, presenting the same training and test sets to all algorithms at each stage. During each test, the training and test sets were normalised by mean-removal and scaling, using the transformation:

$$x_i \leftarrow \frac{(x_i - \overline{x}_i)}{\sigma_i},$$

where x_i is attribute i, $\overline{x}_i$ is the mean of x_i and σ_i is the standard deviation of x_i. Mean and standard deviation were computed using values from the training set only, to prevent indirect bias on test results.

In general, there is no rule to determine ahead of time the optimal number M of prototypes to place in the codebook, so we ran the experiment several times for each data set, with a different value of M each time. Codebooks were initialised by specifying a desired value of M, apportioning that value among the classes in proportion to their representation in the training set, then rounding up to the nearest whole number. The desired values of M were 1, 2, 4, 8, 16, 32 and 64.

The optimal choice of bandwidth parameter σ in LMVQ is also unknown ahead of time, so for each distinct value of M, we did runs with $\sigma = 0.0625$, 0.125, 0.25, 0.5, 1.0, 2.0, 4.0 and 8.0. Previous authors have suggested that a normalised radial basis function network is not very sensitive to the particular value of σ [Bugmann 1998]. For 4 of the data sets (Breast Cancer, Heart Disease, Iris, and Diabetes) this observation holds true, with LMVQ producing very similar results for different values of $\sigma \in \{0.5, 1.0, 2.0, 4.0\}$. However, for the remaining data sets (Glass and Sonar) quite different results emerged, with accuracy proving to be very sensitive to both σ and M.

Classifiers were trained using the following parameter settings. LMVQ was trained with a step size of 0.05, and each training example was presented approximately 20 times, sampling randomly with replacement from the training set. The reference implementations of LVQ were used to construct the LVQ

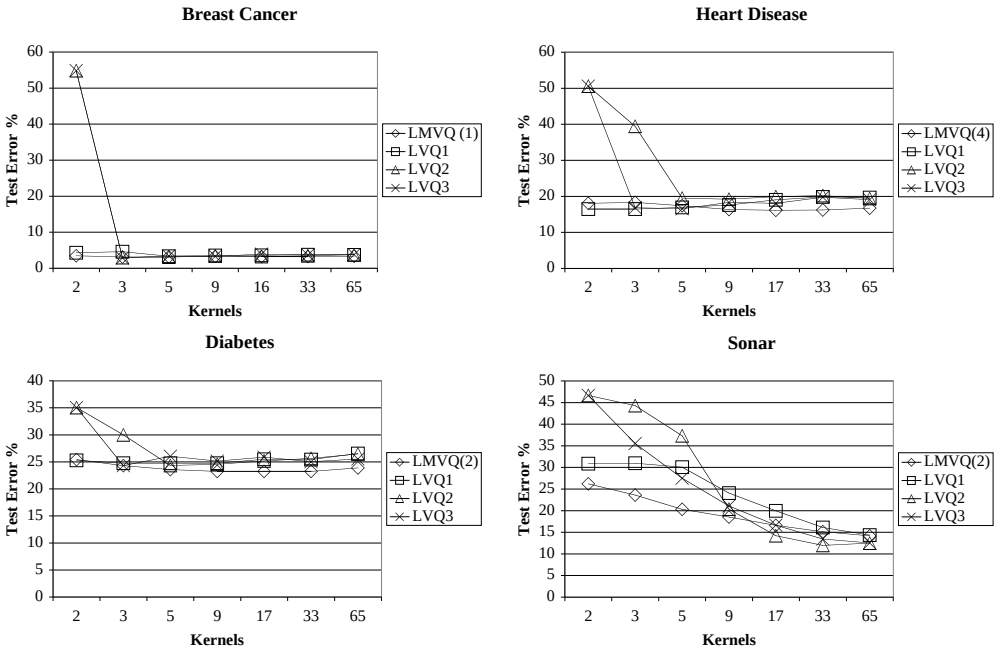


Figure 5: Generalisation error of LMVQ, LVQ1, LVQ2 and LVQ3 algorithms, on a set of 2-class problems. For LMVQ, the value of σ that was used for the test is shown in parentheses.

classifiers, using a two-stage regime. First, the OLVQ1 algorithm was applied, with run length of 10000 presentations, to produce an intermediate codebook. This codebook formed the common starting point for a second pass, which produced three distinct LVQ classifiers, one for each LVQ variant. The series labelled LVQ1 in Figures 5 and 6 were generated by applying OLVQ1 for a further 10000 presentations. The series labelled LVQ2 were produced by applying the LVQ2.1 algorithm for 10000 presentations, with initial learning rate of 0.1 and window size set to 0.3. The series labelled LVQ3 were the result of applying LVQ3 for 10000 presentations, with learning rate of 0.1, window size of 0.3 and ϵ set at 0.5.

Figures 5 and 6 show the average generalisation error for each algorithm, plotted as a function of codebook size M. Although LMVQ was executed with a range of different settings for σ, we show here only the results for the best choice of σ. This value is displayed in parentheses in the legend of the corresponding chart. As noted above, for most of the data sets the results were not very sensitive to the choice of σ.

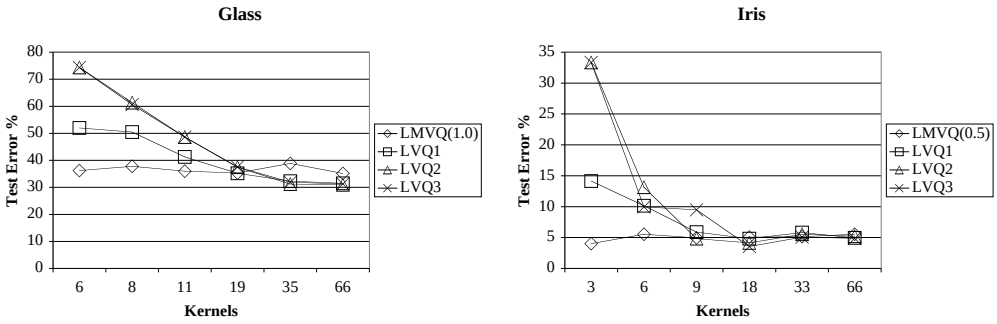


Figure 6: Generalisation error of LMVQ, LVQ1, LVQ2 and LVQ3 algorithms, on problems with more than 2 classes. For LMVQ, the value of σ that was used for the test is shown in parentheses.

Considering the results shown in Figures 5 and 6, we observe at once that all four algorithms approach the same level of accuracy when M is reasonably large. However, when we consider small classifiers with just a few prototypes allocated to each class, we see that LMVQ performs much better than LVQ2.1 and LVQ3 on all data sets, and better than LVQ1 on three of the data sets. On 5 of the 6 datasets, increasing the number of kernels has little impact on the performance of the resulting LMVQ classifier.

These results show that provided a suitable value for σ is chosen, LMVQ can achieve accuracy comparable to that of LVQ, with a smaller number of prototypes. Although the learning steps of LVQ2.1 and LVQ3 are qualitatively similar to that of LMVQ, when the complexity of the codebook is low they are much less effective at mapping the decision surface than LMVQ.

8 Conclusions

This paper describes LMVQ, an on-line training algorithm for a normalised radial basis function classifier. The algorithm seeks to maximise the margin of each training example. We have shown that under certain conditions the behaviour of LMVQ is approximated by the widely known LVQ2.1 and LVQ3 algorithms, vindicating the reasoning upon which these earlier algorithms are founded. The relationship between LMVQ and LVQ is analogous to that between the EM algorithm and the k-means algorithm. As well as this, experimental results indicate that LMVQ is a promising training algorithm in its own right. On an artificial problem LMVQ outperforms major variants of LVQ in terms of classification accuracy and codebook economy. Experiments on some more realistic data sets show that LMVQ is better able to discover a simple decision surface

than LVQ2.1 or LVQ3.

Like the Support Vector Machine (SVM) training algorithm for radial basis function classifiers, LMVQ is based on the idea of increasing the margin over the training set. However the algorithm itself is quite different from SVM learning, in which a pool of prototypes is selected and the classifier is adapted by changing the mixing weights of the associated gaussian kernels. LMVQ holds the mixing weights constant but adjusts the locations of the prototypes to model the decision surface.

A number of questions remain unanswered about the LMVQ algorithm. Since the choice of bandwidth parameter determines the level of resolution at which the decision surface may be mapped, it will be highly desirable to devise an efficient way to set this parameter. One possible approach is to attempt to learn the best value of σ directly from the training data. However, our early attempts to use gradient learning for σ have been quite discouraging. Another approach would be to use σ as an adaptive control, allowing models to be built by progressively adding more detailed features. Another key question that must be answered for all algorithms of this type is how many kernels should be incorporated in the codebook. Although we have implemented LMVQ with gaussian kernels, any suitable basis function could be used, opening a further avenue for enquiry. Theoretical or empirical evidence that LMVQ does indeed cause a large proportion of the training set to be classified correctly with large margin would be highly desirable. Also, convergence and stability this algorithm have yet to be analysed.

References

[Boser, Guyon & Vapnik 1992] Boser, B.E., Guyon, I.M., & Vapnik, V.N.: A Training Algorithm for Optimal Margin Classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* 144–152.

[Breiman 1997] Breiman, L.: Arcing the edge. Technical report, Statistics Department, U. C. Berkeley.

[Bugmann 1998] Bugmann, G.: Normalized Gaussian Radial Basis Function Networks. *Neurocomputing* **20** 97–110

[Cover & Hart 1967] Cover, T. M., Hart P. E.: Nearest neighbour pattern classification. *IEEE Transactions on Information Theory* **13** 21–27

[Friedman *et al* 1998] Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Stanford University.

[Freund & Schapire 1997] Freund, Y., Schapire, R. E.: A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* **55** 119–139

[Geva & Sitte 1991a] Geva, S. & Sitte, J.: Adaptive Nearest Neighbour Pattern Classification. *IEEE Transactions on Neural Networks* **2** 318–322

[Geva & Sitte 1991b] Geva, S. & Sitte, J.: An exponential response neural net. *Neural Computation* **3** 623-632

[Jordan & Jacobs 1991] Jordan, M. I., Jacobs, R. A.: Hierarchical mixtures of experts and the EM algorithm. *Neural Computation* **6** 181–214

[Jordan & Xu 1995] Jordan, M. I., Xu, L: Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks* **8** 1409–1431

[Kohonen 1997] Kohonen, T.: *Self-Organizing Maps – 2nd ed.* New York, NY: Springer-Verlag

[Mason *et al.* 1999] Mason, L., Baxter J., Bartlett, P. L., Frean, M.: Boosting algorithms as gradient descent in function space. Technical report, Research School of Information Sciences and Engineering, Australian National University.

[Murphy & Aha 1994] Murphy, P. & Aha, D. *UCI repository of machine learning databases – a machine-readable data repository.* Maintained at the Department of Information and Computer Science, University of California, Irvine. Anonymous FTP from ics.uci.edu in directory pub/machine-learning-databases.

[Shapire *et al.* 1998] Shapire, R., Freund, Y., Bartlett, P. L., Lee, W.: Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics* **26** 1651–1686

[Shawe-Taylor *et al.* 1996] Shawe-Taylor, J., Bartlett, P. L., Williamson, R. C., Anthony, M.: A framework for structural risk minimisation. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory* 68–76

[Vapnik 1995] Vladimir N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag.